Semester-Long Internship Report

On

DevOps Intern

Submitted by

Shiv Chaudhary

Under the guidance of

Lee Thomas Stephen
FOSSEE, IIT Bombay

and mentor

Mr. Rohan Mhatre

July 25, 2024

About FOSSEE Project:

The FOSSEE project is part of the National Mission on Education through ICT, with the thrust area being "Adaptation and deployment of open source simulation packages equivalent to proprietary software, funded by MHRD, based at the Indian Institute of Technology Bombay (IITB).

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes using FLOSS tools in academia and research. The FOSSEE project is part of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education(MoE), Government of India.

About The National Mission on Education:

To improve the levels of education in India, the Ministry of Human Resource Development has launched an ambitious educational mission with an outlay of about US $ One billion. It is proposed that it be implemented through Information and Communication Technologies. The following minimum requirements are placed to fund a project through this mission:

- It has to be inter-institutional.
- It should be development-oriented in any general field of college-level education.
- Any material developed through this mission has to be delivered as open source.
- It should belong to any one of the about twenty sub-missions identified in the mission document, available at www.sakshat.ac.in

To know more about FOSSEE and its projects, visit: **https://fossee.in/**

Acknowledgment

I would like to express my deepest gratitude to everyone who contributed to successfully completing my internship. First and foremost, I am profoundly thankful to my Lee Thomas Stephen and mentor, Mr. Rohan Mhatre, for their invaluable guidance, support, and encouragement throughout this journey. Their expertise, patience, and willingness to share their knowledge have been instrumental in my learning and growth. They gave me the tools and confidence to tackle challenges and excel in my project.

Furthermore, I am grateful to the FOSSEE organization for providing me with this wonderful opportunity. This experience has been pivotal in shaping my career aspirations and has given me a clearer understanding of the professional world.

I would also like to acknowledge my professors and academic advisors for their continuous support and guidance, which laid the foundation for my internship. Their encouragement and advice have been invaluable throughout this process.

Lastly, I extend my heartfelt appreciation to my family and friends for their unwavering support and encouragement. Their belief in my abilities and their constant motivation has been a source of strength for me. This internship has been an incredibly enriching experience, and I am grateful to all who made it possible.

Thank you all for being a part of this significant phase of my professional journey.

Contents

Chapter 1

Introduction

Google Cloud consists of a set of physical assets, such as computers and hard disk drives, and virtual resources, such as virtual machines (VMs), that are contained in data centers around the globe. Each data center location is in a *region*. Regions are available in Asia, Australia, Europe, Africa, the Middle East, North America, and South America. Each region is a collection of *zones* that are isolated from each other within the region. Each zone is identified by a name that combines a letter identifier with the name of the region. For example, zone a in the East Asia region is named asia-east1-a.

This distribution of resources provides several benefits, including redundancy in case of failure and reduced latency by locating resources closer to clients. This distribution also introduces some rules about how resources can be used together.

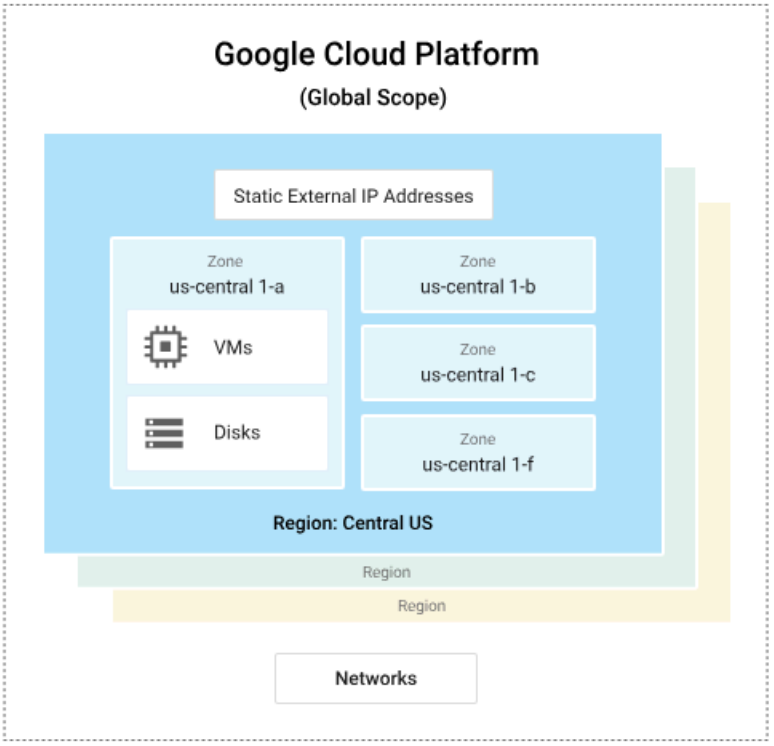
Accessing resources through services

In cloud computing, what you might be used to thinking of as software and hardware products become *services*. These services provide access to the underlying resources. The list of available Google Cloud services is long, and it keeps growing. When you develop your website or application on Google Cloud, you mix and match these services into combinations that provide the infrastructure you need and then add your code to enable the scenarios you want to build.

Global, regional, and zonal resources

Some resources can be accessed by any other resource across regions and zones. These *global resources* include pre-configured disk images, disk snapshots, and networks. Some resources can be accessed only by resources that are located in the same region. These *regional resources* include static external IP addresses. Other resources can be accessed only by resources that are located in the same zone. These *zonal resources* include VM instances, their types, and disks.

The following diagram shows the relationship between global scope, regions and zones, and some of their resources:



Projects

Any Google Cloud resources that you allocate and use must belong to a project. You can think of a project as the organizing entity for what you're building. A project is made up of the settings, permissions, and other metadata that describe your applications. Resources within a single project can work together easily, for example, by communicating through an internal network, subject to the regions-and-zones rules. A project can't access another project's resources unless you use Shared VPC or VPC Network Peering.

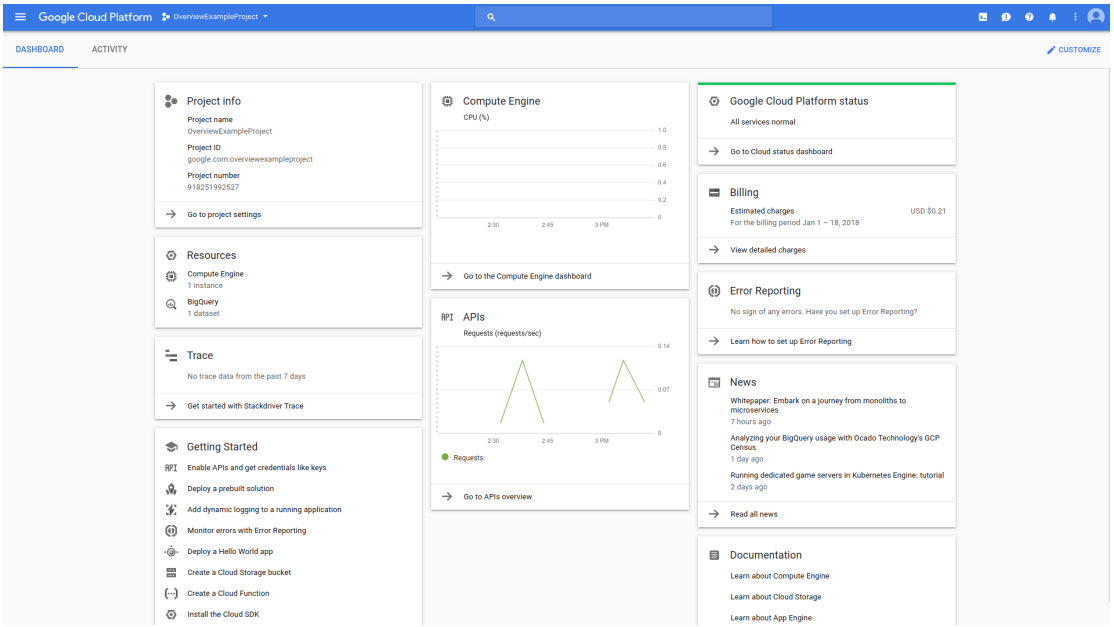
Each Google Cloud project has the following:

- A project name, which you provide.
- A project ID, which you can provide, or Google Cloud can provide it for you.
- A project number, which Google Cloud provides.

Each project ID is unique across Google Cloud. After you have created a project, you can delete the project but its ID can never be used again. You can create multiple projects and use them to separate your work in whatever way makes sense for you. For example, you might have one project that can be accessed by all team members and a separate project that can only be accessed by certain team members.

Ways to interact with the services

Google Cloud console



The Google Cloud console provides a web-based, graphical user interface that you can use to manage your Google Cloud projects and resources. When you use the Google Cloud console, you either create a new project or choose an existing project and then use the resources that you create in the context of that project.

Command-line interface

If you prefer to work at the command line, you can perform most Google Cloud tasks by using the Google Cloud CLI. The gcloud CLI lets you manage development workflow and Google Cloud resources in a terminal window.

For example, you can create a Compute Engine virtual machine (VM) instance by running the gcloud compute instances create command in the shell environment.

For a list of gcloud commands, see the gcloud reference.

For more information about Cloud Shell, see How Cloud Shell works.

Firewall rules

Each VPC network implements a distributed virtual firewall that you can configure. Firewall rules allow you to control which packets are allowed to travel to which destinations. Every VPC network has two implied firewall rules that block all incoming connections and allow all outgoing connections.

The default network has additional firewall rules, including the default-allow-internal rule, which permits communication among instances in the network.

Read more about firewall rules.

IP addresses

Google Cloud resources, such as Compute Engine VM instances, forwarding rules, GKE containers, and App Engine, rely on IP addresses to communicate.

Read more about IP addresses.

Private Google Access

When you enable Private Google Access for a subnet, instances in a subnet of a VPC network can communicate with Google APIs and services by using private IP addresses instead of external IP addresses.

Chapter 2

Google Cloud Platform (GCP) Setup and Configuration

2.1 Overview

The setup process includes the following phases:

1. **Establish your organization, administrators, and billing**: Set up the top-level node of your hierarchy, create initial administrator users, and connect your payment method.
2. **Create an initial architecture**: Select an initial folder and project structure, assign access, configure logging, apply security settings, and set up your network.
3. **Deploy your settings**: Your initial architecture choices are compiled in Terraform configuration files. You can quickly deploy through the Google Cloud console or download the files to customize and iterate using your own workflow.
4. **Apply monitoring and support settings**: Apply recommended monitoring and support settings to bolster your architecture.

VPC networks

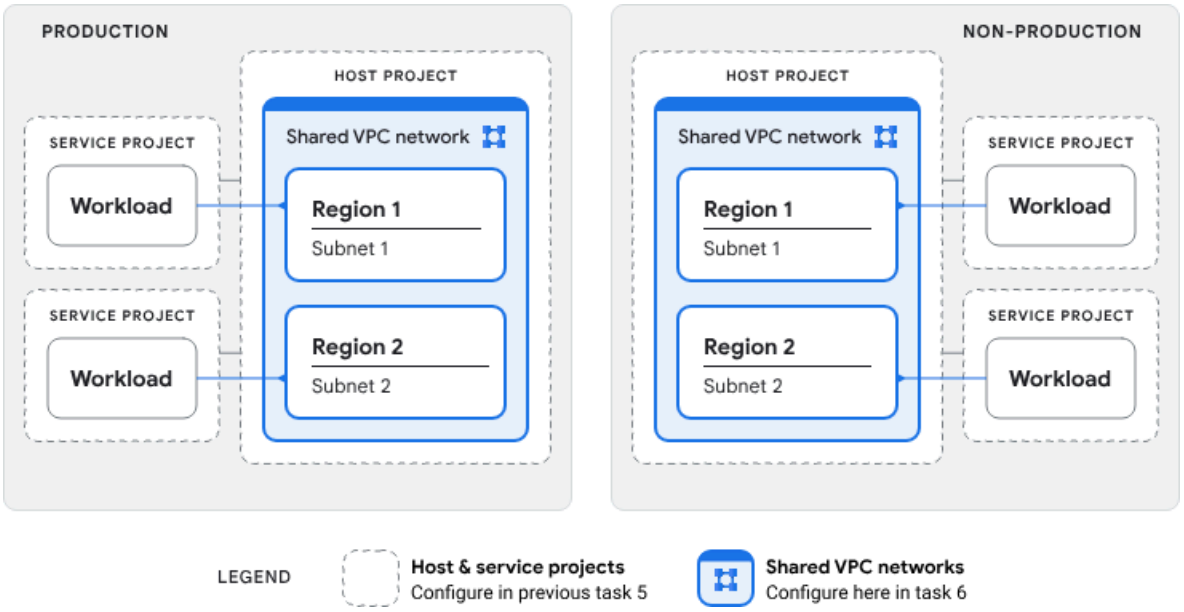
In this task, you set up your initial networking configuration, which you can scale as your needs change.

Virtual Private Cloud architecture

A Virtual Private Cloud (VPC) network is a virtual version of a physical network that is implemented inside of Google's production network. A VPC network is a global resource that consists of regional subnetworks (subnets).

VPC networks provide networking capabilities to your Google Cloud resources such as Compute Engine virtual machine instances, GKE containers, and App Engine flexible environment instances.

Shared VPC connects resources from multiple projects to a common VPC network so that they can communicate with each other using the network's internal IP addresses. The following diagram shows the basic architecture of a Shared VPC network with attached service projects.



When you use Shared VPC, you designate a host project and attach one or more service projects to it. Virtual Private Cloud networks in the host project are called Shared VPC networks.

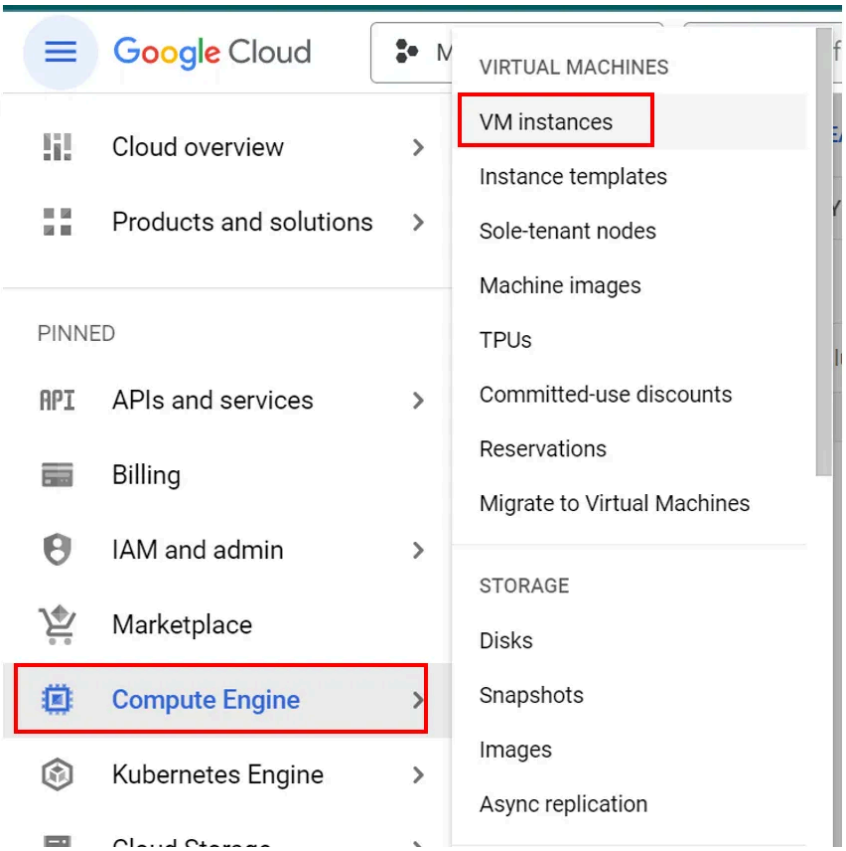
The example diagram shows production and non-production host projects, each of which contains a shared VPC network. You can use a host project to centrally manage the following:

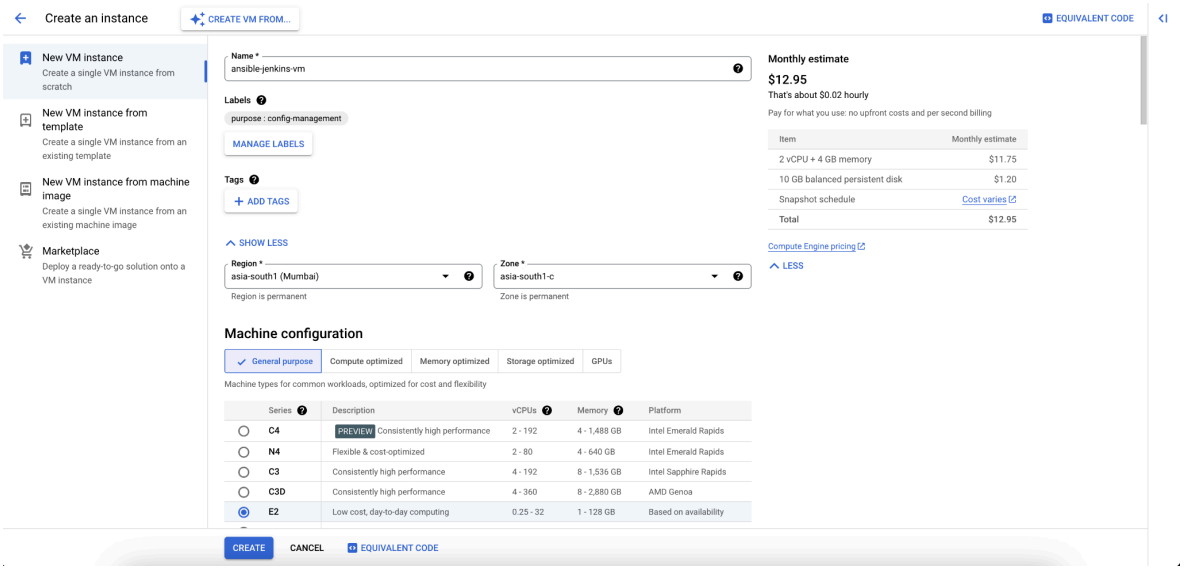
- Routes
- Firewalls
- VPN connections
- Subnets

2.2 Setting up Ansible and Jenkins on a Virtual Machine

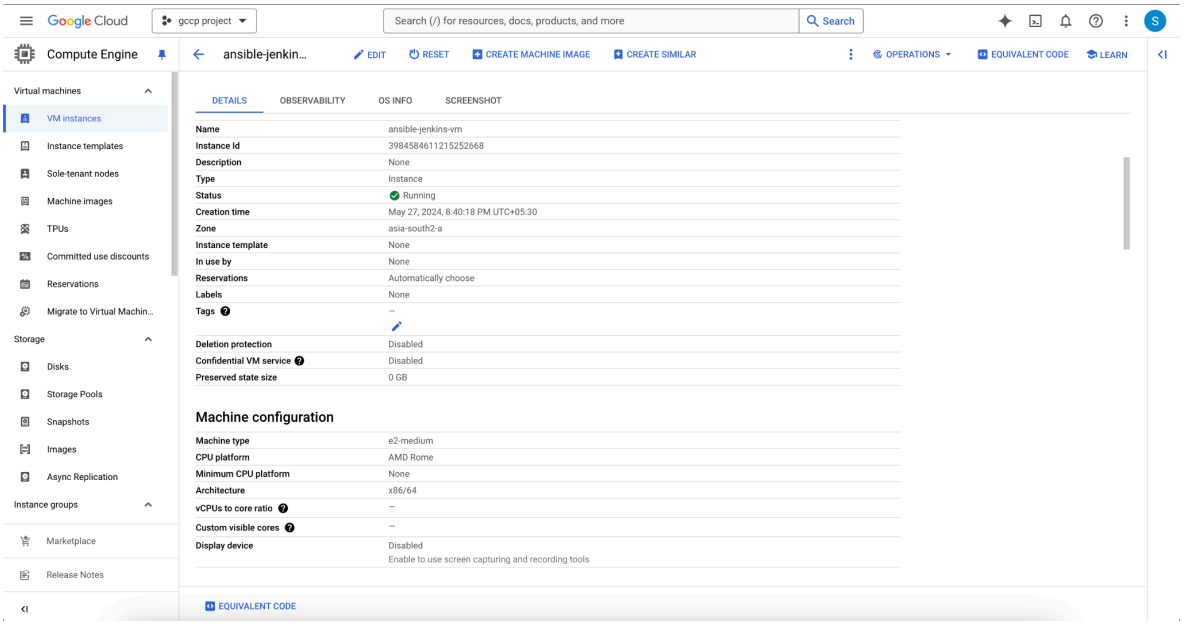
2.2.1 Installation

- Login to the GCP account
- Create 2 VMs in GCP. 1- ansible-jenkins-vm 2- app-vm
- To create a VM follow the below steps

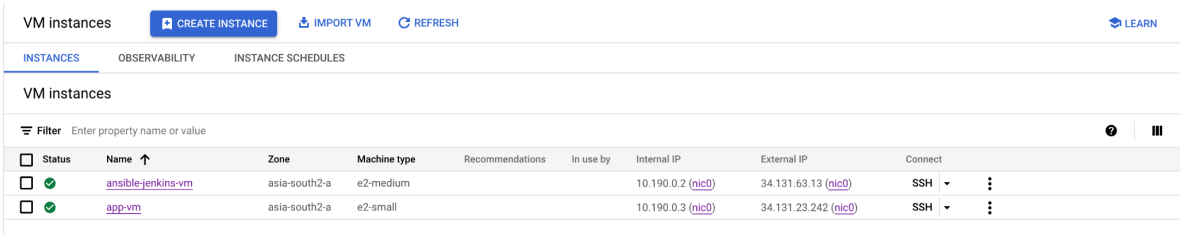




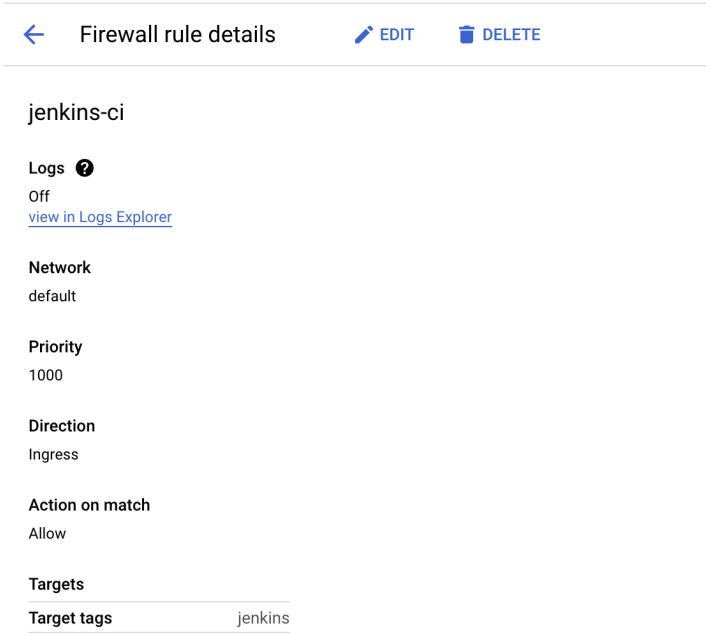
Vm is Created Successfully

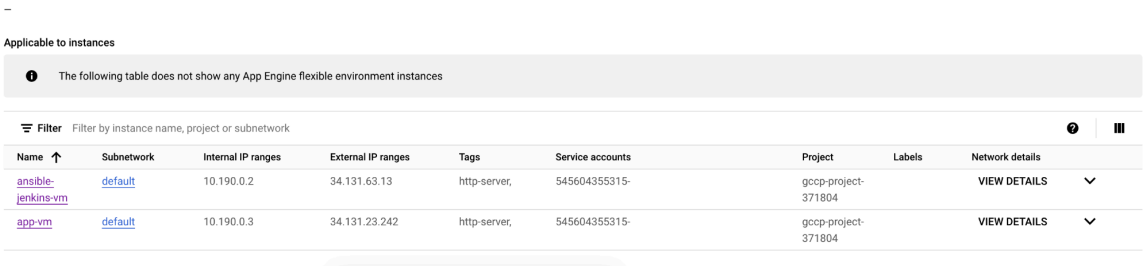


Follow the same steps and create 1 more instance like this and name it app-vm



Firewall Rules:





ansible-jenkins-vm

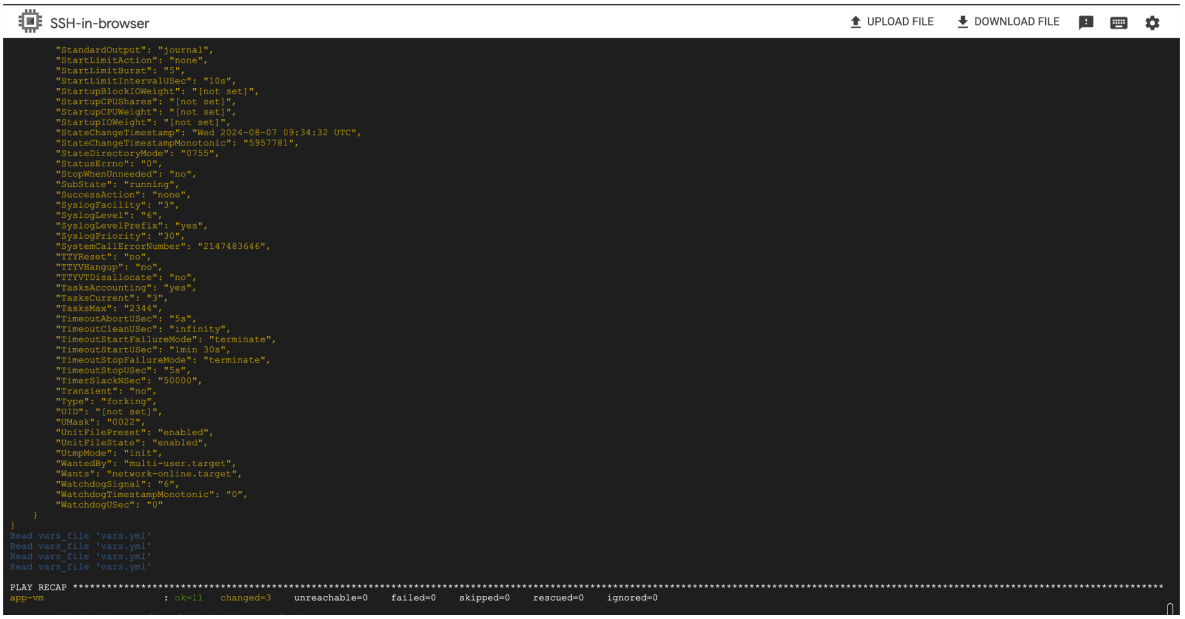
```
shivchaudharytemp@ansible-jenkins-vm:~$ jenkins --version
2.462.1
shivchaudharytemp@ansible-jenkins-vm:~$ ansible --version
ansible [core 2.14.3]
  config file = None
  configured module search path = ['/home/shivchaudharytemp/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/shivchaudharytemp/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.11.2 (main, May  2 2024, 11:59:08) [GCC 12.2.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
shivchaudharytemp@ansible-jenkins-vm:~$
```

The above Image Shows the Version of Ansible and Jenkins installed on the **ansible-jenkins-vm** VM.

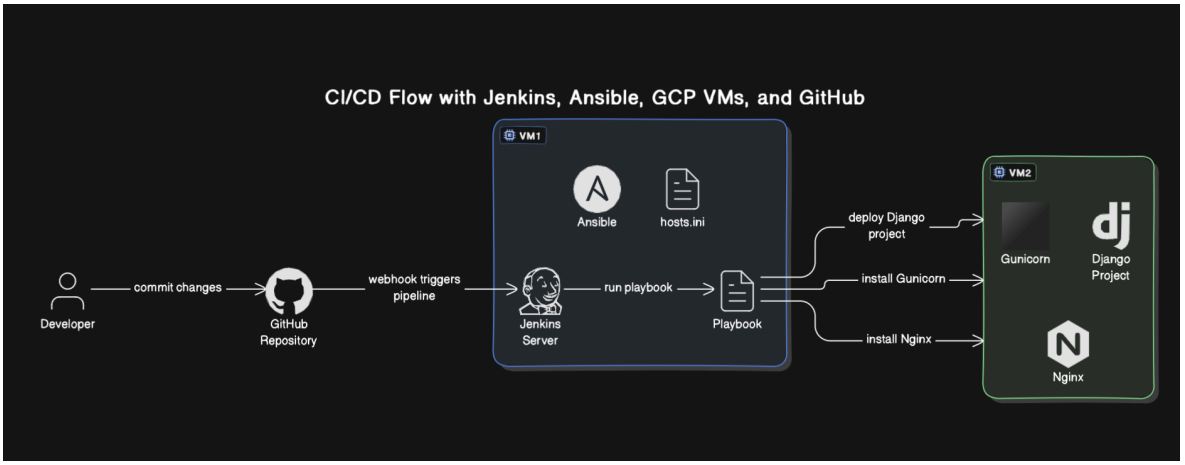
```
Linux ansible-jenkins-vm.asia-south2-a.c.gccp-project-371804.internal 6.1.0-23-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.99-1 (2024-07-15) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Aug 16 22:29:40 2024 from 35.235.244.192
shivchaudharytemp@ansible-jenkins-vm:~$ ls
ansible-playbooks  deploy_django_app.yml  hosts  nginx.conf.j2  p.yaml  playbook.yaml  vars.yml
shivchaudharytemp@ansible-jenkins-vm:~$ cat nginx.conf.j2
server {
    listen 80;
    server_name 34.131.218.104;
    location = /favicon.ico { access_log off; log_not_found off; }
        root /var/www/sysad_intern;

    location / {
        include proxy_params;
        proxy_pass http://unix:/var/www/sysad_intern/sysad_intern.sock;
    }
}
shivchaudharytemp@ansible-jenkins-vm:~$ cat playbook.yaml
---
- name: Deploy Django Application
  hosts: app_vms
  become: yes
  vars_files:
    - vars.yml
  tasks:
    - name: Install required packages
      apt:
        name: "{{ item }}"
        state: present
        update_cache: yes
      with_items:
        - git
        - python3-pip
        - python3-venv
        - nginx
        - virtualenv

    - name: Clone the Django application repository
      git:
        repo: "{{ django_app_repo }}"
        dest: "{{ app_dir }}"
        version: "main"

    - name: Create a Python virtual environment
      command: python3 -m venv {{ app_dir }}/venv
      args:
        creates: "{{ app_dir }}/venv"

    - name: Install Python dependencies
      pip:
        requirements: "{{ app_dir }}/requirements.txt"
```

The above Images show configuration files and the playbook used by Ansible for the provision of the Django app on **app-vm.**

Ansible playbook runs perfectly as desired and creates configuration on **app-vm.**



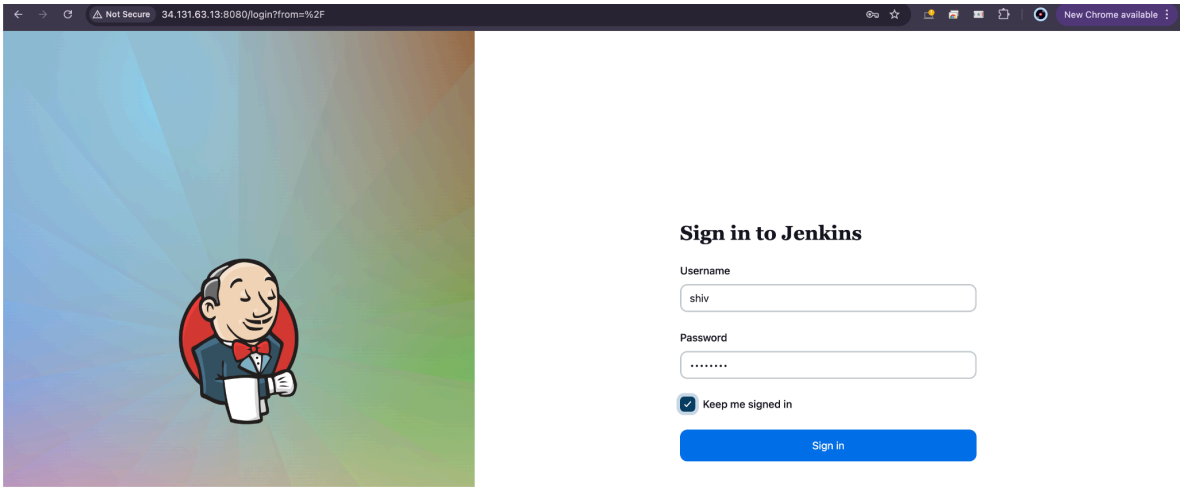
Architecture :



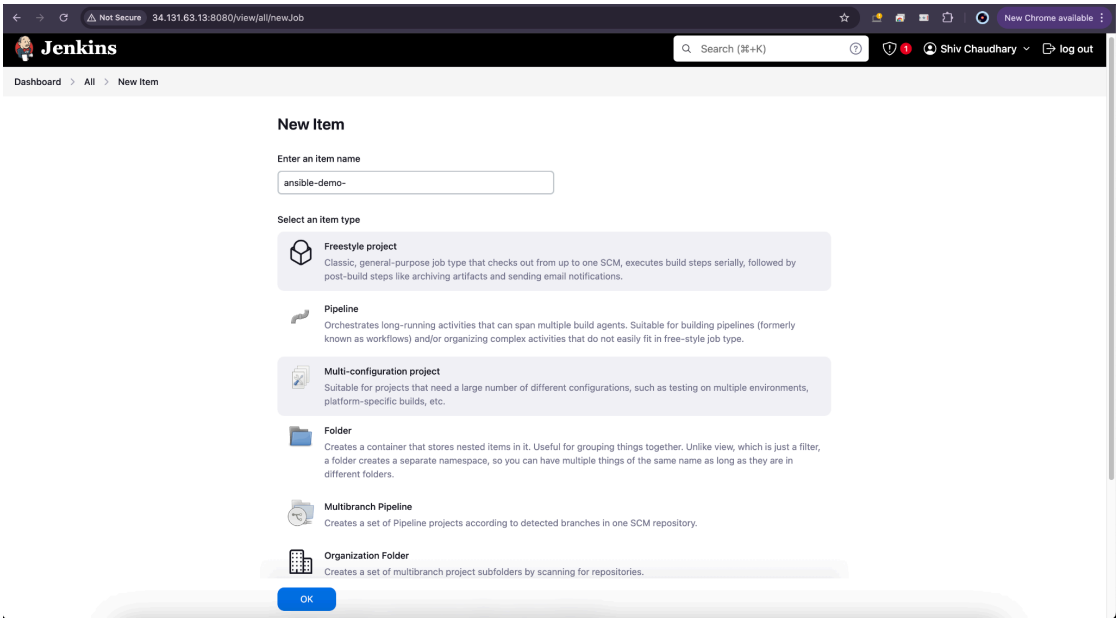
Jenkins is an open-source continuous integration/continuous delivery and deployment (CI/CD) automation software DevOps tool. It is used to implement CI/CD workflows called pipelines.

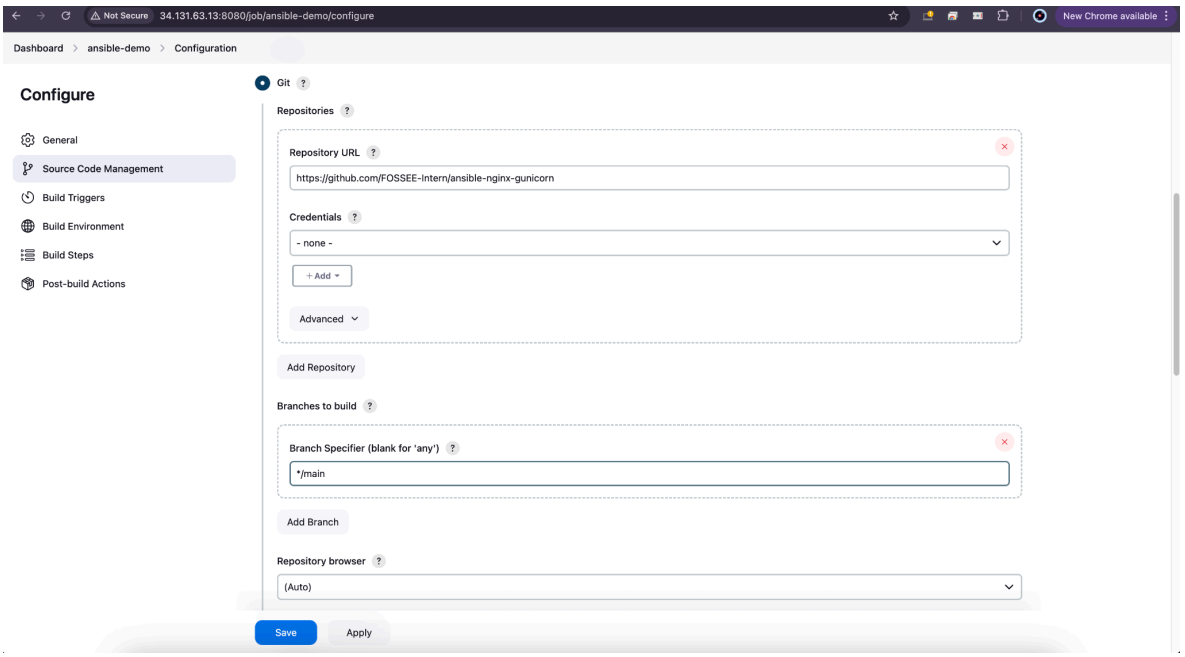
Ansible is an open-source, command-line IT automation software application. It can configure systems, deploy software, and orchestrate advanced workflows to support application deployment, system updates, and more.

Home Page of Jenkins running on **ansible-jenkins-vm** vm using external ip on port 8080.

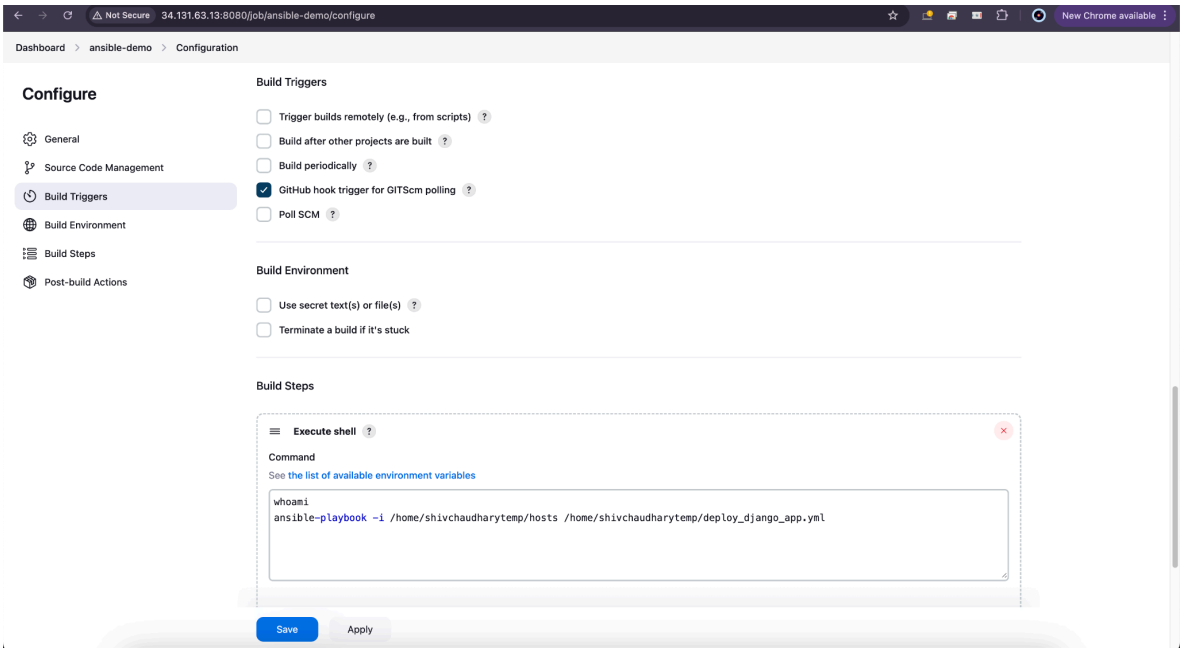


Creating a project

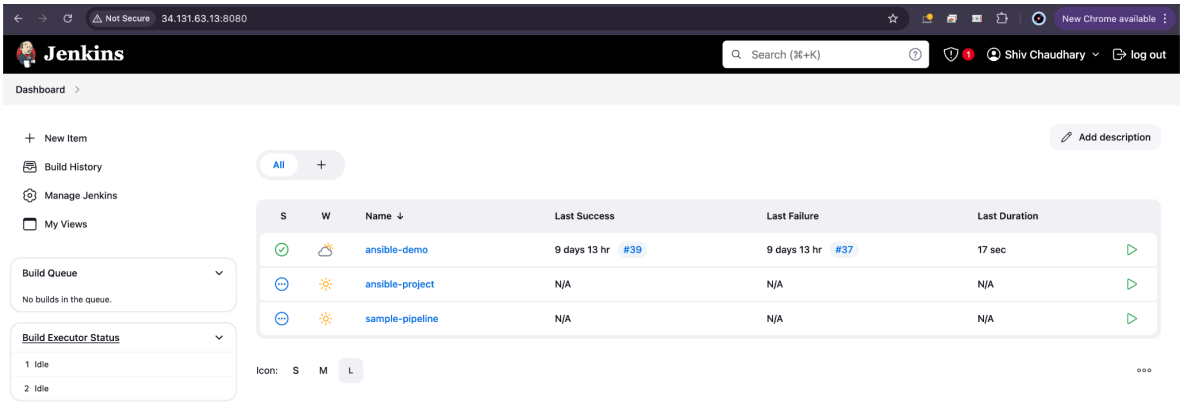




Configure jenkins for pipeline:

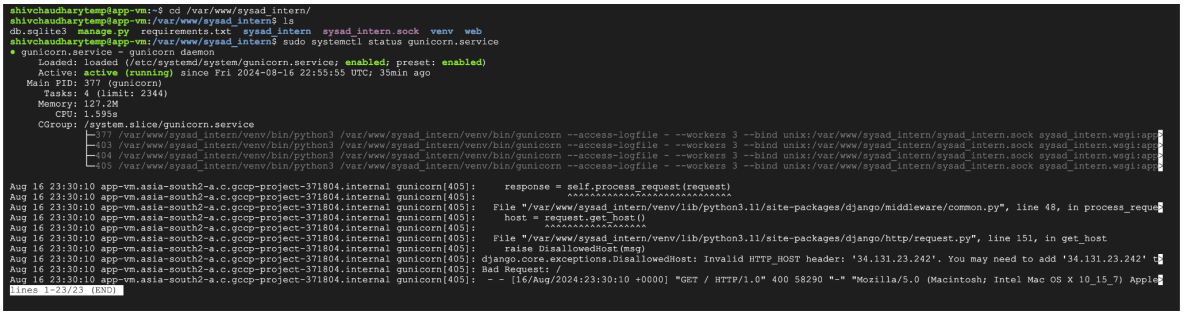


Jenkins Configuration



app-vm

py app config is done successfully in app-vm from ansible-jenkins-vm using ansible.



2.2.2 Summary

This section documents the process of deploying a Python application on a virtual machine (app-vm) using Ansible, orchestrated from another virtual machine (ansible-jenkins-vm) on the Google Cloud Platform (GCP). The deployment process also involves setting up a continuous integration/continuous deployment (CI/CD) pipeline using Jenkins, which triggers the deployment whenever new code is pushed to GitHub.

Setup and Configuration

1. Infrastructure Preparation:
   Both app-vm and ansible-jenkins-vm were provisioned on GCP. The app-vm serves as the host for the Python application, while the ansible-jenkins-vm acts as the control node for managing configurations and deployments using Ansible.
2. Ansible Installation and Configuration:
   Ansible was installed on ansible-jenkins-vm, and the necessary SSH key pair was generated for secure communication between the two VMs. The public SSH key was added to the authorized_keys file on app-vm to enable passwordless authentication.
3. Playbook Development:
   A custom Ansible playbook was created to automate the deployment process. The playbook included tasks for:
   - Installing Dependencies: Ensuring that Python, pip, and other required packages were installed on app-vm.
   - Cloning the Repository: Pulling the latest version of the application code from GitHub.
   - Setting Up Virtual Environment: Creating and activating a Python virtual environment on app-vm.
   - Installing Python Requirements: Installing all necessary Python packages using the requirements.txt file.
   - Configuring Nginx: Deploying and configuring Nginx as a reverse proxy to serve the Python application. A Jinja2 template was used to dynamically generate the Nginx configuration file, tailored to the environment on app-vm.
   - Starting Gunicorn: Ensuring that Gunicorn was set up as the application server to serve the Python application.
4. Nginx Configuration using Jinja2:
   The Nginx configuration was templated using Jinja2 within the Ansible playbook. This allowed for the dynamic insertion of variables such as server names and port numbers, making the deployment flexible and adaptable to different environments.

5. Jenkins CI/CD Pipeline:
   Jenkins was installed on ansible-jenkins-vm to facilitate continuous deployment. A Jenkins pipeline was configured to trigger the Ansible playbook whenever new code is pushed to the GitHub repository. The pipeline stages included:
   - Code Checkout: Pulling the latest code from the repository.
   - Ansible Playbook Execution: Running the Ansible playbook to deploy the updated code on app-vm.
   - Post-Deployment Tests: Running basic tests to ensure the application is running smoothly after deployment.

Workflow

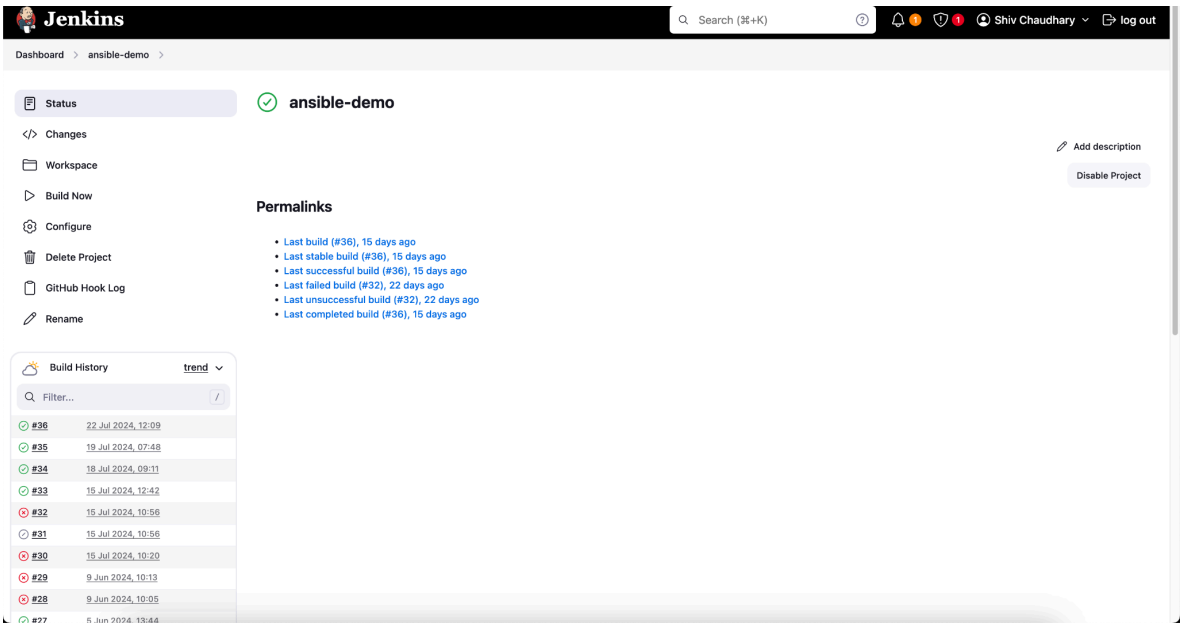
1. A developer pushes new code to the GitHub repository.
2. Jenkins detects the push event and triggers the CI/CD pipeline.
3. Jenkins executes the Ansible playbook on ansible-jenkins-vm.
4. The playbook performs the necessary tasks on app-vm to deploy the updated Python application.
5. The application is served via Nginx on app-vm, with Gunicorn as the application server.

Challenges and Solutions

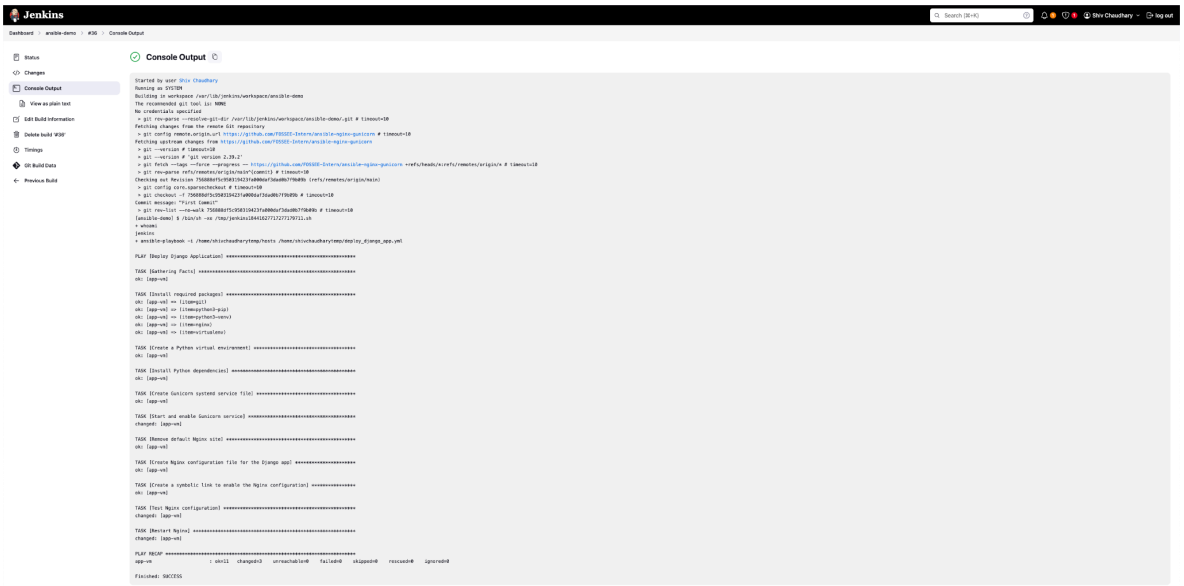
- SSH Connectivity: Initial challenges were faced in establishing SSH connectivity between the two VMs. This was resolved by correctly setting up the SSH keys and ensuring that the necessary ports were open on both VMs.
- Nginx Configuration: The Nginx configuration required careful templating to ensure that the application was correctly served. This was managed by using Jinja2 templates within the Ansible playbook.
- Jenkins Integration: Integrating Jenkins with the Ansible playbook required fine-tuning to ensure that the deployment process was seamless and triggered automatically upon code updates.

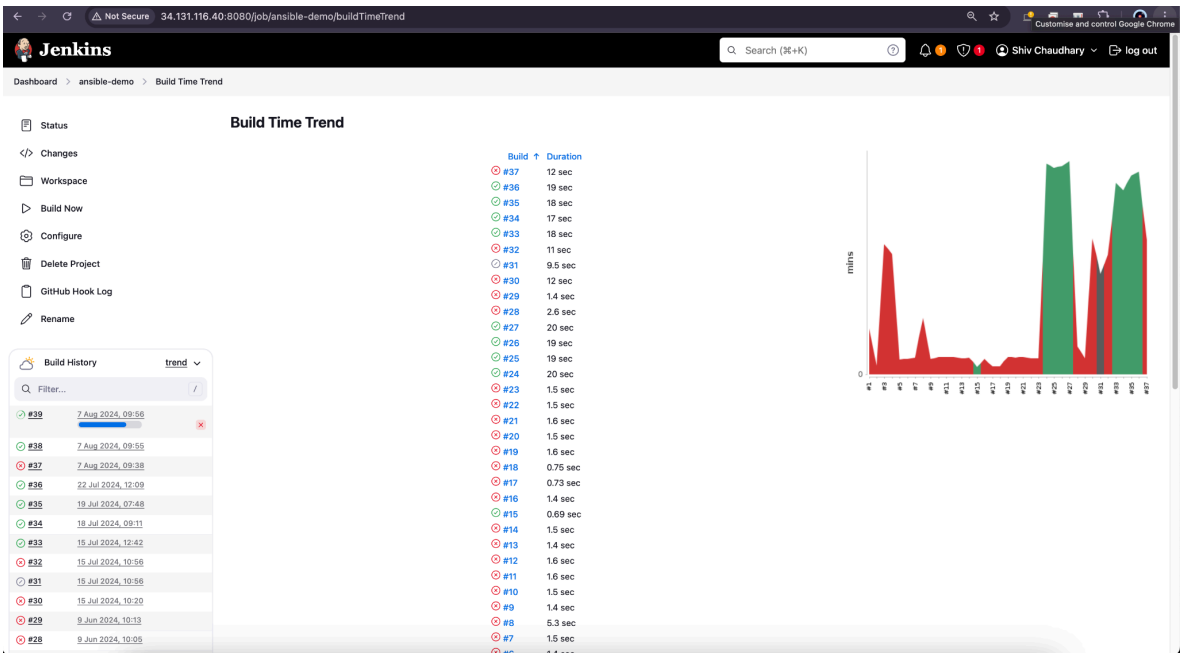
2.2.3 Testing

Pipeline is running successfully when pushing code to github.



Console Output





App is running as desired.

Test About Page

Chapter 4

Learnings

Technical Skills Enhancement

- **Gained Proficiency in GCP and DevOps Tools**: Developed a strong understanding of Google Cloud Platform (GCP), including setting up and managing virtual machines, configuring network settings, and deploying applications.
- **Mastered Ansible and Jenkins**: Acquired advanced skills in using Ansible to automate the deployment process and Jenkins to set up CI/CD pipelines, ensuring seamless integration and delivery.
- **Enhanced Scripting and Automation**: Improved my ability to write and troubleshoot Ansible playbooks and Jenkins pipelines, optimizing the deployment of Python applications across multiple VMs.
- **Advanced Troubleshooting Skills**: Developed the ability to identify and resolve issues in deployment processes, including SSH connectivity, Nginx configuration, and pipeline failures.

Feedback and Continuous Improvement.

- **Adopted a Mindset of Continuous Learning**: Embraced the importance of staying updated with the latest tools and practices in DevOps and cloud computing, continuously seeking ways to enhance my skill set.
- **Self-Assessment and Growth**: Regularly assessed my progress in mastering new technologies and sought feedback to identify areas for further improvement.

Communication Skills

- **Improved Technical Communication**: Enhanced my ability to clearly document complex processes, such as setting up Ansible and Jenkins, making the information accessible to team members and future users.
- **Collaborative Problem-Solving**: Gained experience in effectively communicating with team members to troubleshoot issues, share knowledge, and collaboratively improve workflows.
- **Guiding Peers**: Developed the ability to explain DevOps concepts, such as CI/CD pipelines and cloud infrastructure, to colleagues, contributing to a collaborative learning environment

Chapter 5

Conclusion

My internship experience has been profoundly enriching and has significantly contributed to my professional and personal growth. Engaging in the development of the FOSSEE System Administration allowed me to apply knowledge in a practical setting, enhancing my skills in DevOps and problem-solving in general. The challenges I encountered and overcame during these projects sharpened my problem-solving skills and provided a deeper understanding of working on open-source projects in general.

Working on various tasks such as astro-ansible-gcp and drupal-mariadb-gcp enhanced my technical skills. This not only expanded my knowledge of system administration but also taught me the importance of comprehensive documentation and testing. On the other hand, tools like Vagrant provided me an opportunity to learn about local testing. This internship has helped outline my career aspirations. The hands-on experience has prepared me for future professional challenges. I am deeply grateful for this opportunity and confident that the skills and insights gained during this internship will be immensely beneficial to my career development. As I move forward, I am excited to leverage the knowledge and experience acquired to contribute meaningfully to the tech industry, especially the open-source world, and achieve my professional goals.

Projects:

1. Ansible-Jenkins-GCP:

- **Project Title:** Ansible-Jenkins-GCP
- **Overview:** This project establishes a CI/CD pipeline that leverages Jenkins to automate the deployment of a Django application on virtual machines. The integration with GitHub is achieved through webhooks, while Ansible simplifies the creation and deployment processes.
- **DevOps Principles:** Employed Infrastructure as Code (IaC) and CI/CD methodologies.
- **Challenges Faced:**
    - Mastering Ansible's unique syntax and diverse modules.
    - Ensuring playbooks could be reused without unintended side effects.
    - Gaining a comprehensive understanding of Jenkins' functionalities with GCP and Ansible.
    - Safeguarding sensitive environmental variables to prevent exposure.
    - Learning Nginx configuration best practices for application hosting.
- **Solutions Implemented:**
    - Incorporated shell scripts and systemd services to enhance playbook idempotence.
    - Utilized Jenkins' built-in options to securely manage environmental variables.
- **Key Takeaways:** Acquired practical experience in writing Ansible playbooks and managing server configurations, along with an understanding of the advantages of using tools like Jenkins and GitHub Actions.

2. Astro-Ansible-GCP:

- **Project Title:** ASTRO-ANSIBLE-GCP
- **Overview:** This project automates the deployment of an Astro website on GCP by implementing a Jenkins and Ansible pipeline. A GitHub webhook triggers the Jenkins pipeline upon detecting new commits, leading to the execution of Ansible playbooks for VM instance provisioning and website deployment.
- **DevOps Principles:** Focused on IaC alongside CI/CD practices.
- **Challenges Faced:**
    - Developing a strategy for maintaining idempotent Ansible playbooks.
    - Navigating the learning curve of integrating Jenkins with GCP.
    - Understanding the nuances of Nginx configurations to effectively serve the application.
- **Solutions Implemented:**
    - Used Jenkins' internal features for secure environmental variable management.
- **Key Takeaways:** Gained hands-on experience in setting up automated CI/CD pipelines and managing GCP infrastructure while also enhancing skills in scripting, version control, and workflow automation.

3. DRUPAL-MARIADB-GCP-ANSIBLE:

- **Project Title:** DRUPAL-MARIADB-GCP-ANSIBLE
- **Overview:** This project automates the deployment of a Drupal website using Jenkins and Ansible on a GCP VM instance. The setup includes configuring Jenkins to respond to GitHub webhooks, utilizing Ansible playbooks for VM provisioning, and the installation of necessary software dependencies.
- **DevOps Principles:** Implemented IaC with Jenkins, CI through GitHub, and continuous deployment to GCP.
- **Challenges Faced:**
    - Ensure that Composer installations are set on an appropriate path for accessibility.
    - Managing the state of the Drupal directory to ensure it was both created and clear for new deployments.
    - Effectively reloading the PHP-FPM and Nginx services to apply changes without downtime.
- **Solutions Implemented:**
    - Utilized the Composer module for streamlined installation.
    - Configured PHP-FPM to use a port instead of a socket for better accessibility.
- **Key Takeaways:** Learned to set up Jenkins on GCP and automate the deployment of Drupal, gaining valuable insights into managing environment variables securely and leveraging GitHub webhooks for automation.

4. DRUPAL-MARIADB-GCP-ANSIBLE-UPDATE:

- **Project Title:** DRUPAL-MARIADB-GCP-ANSIBLE-UPDATE
- **Overview:** This script updates an existing Drupal environment, ensuring both the Drupal core and its modules are brought up to the latest versions.
- **DevOps Principles:** Applied IaC principles using Jenkins and Ansible for maintenance tasks.
- **Challenges Faced:**
    - Ensuring compatibility of the existing modules with the latest Drupal version.
    - Implementing a rollback mechanism in case the update process encounters issues.
    - Coordinating updates without affecting the live environment or user experience.
- **Key Takeaways:** Developed skills in automating Drupal updates through Jenkins and Ansible while also enhancing my understanding of securely managing GCP credentials and automating deployment processes via GitHub webhooks.

Reference

- Ansible
- Jenkins
- GCP Cloud
- GitHub Code Link